An Approach Based on Edge Coloring of Tripartite Graph for Designing Parallel LDPC Interleaver Architecture

Awais SANI, Philippe COUSSY, Cyrille CHAVET, Eric MARTIN.
Lab-STICC, Université de Bretagne-Sud, Lorient

***Abstract*- A practical and feasible solution for LDPC decoder is to design partially-parallel hardware architecture. These architectures are efficient in terms of area, cost, flexibility and performances. However, this type of architecture is complex to design since concurrent read and write accesses to data have to be performed at each time instance without any conflict. To solve this memory mapping problem, we present in this paper, an original approach based on a tripartite graph modeling and a modified edge coloring algorithm to design parallel LDPC interleaver architecture.**

1. INTRODUCTION

Low-density parity-check (LDPC) codes [1] has gained a lot of attention in information theory community thanks to their near Shannon limit error correction capabilities and the explicit parallelism exhibited by their iterative decoding algorithm. These codes have already been included in several wireless communication standards such as DVB-S2 and DVB-T2 [3], WiFi (IEEE 802.11n) [4] or WiMAX (IEEE 802.16e) [5].

LDPC codes are linear block codes and are represented either by a sparse parity check matrix H or by a bipartite graph which is called Tanner graph [2]. Figure 1.a shows the tanner graph of a LDPC code, which consists of two sets of vertices: variable node set (VN) and check node set (CN). A data $v_i \in$ VN represents one bit in the codeword (i.e. data to be processed) whereas $c_j \in$ CN represents a check equation used in generating parity check bits (i.e. operation to be done on the data). A v_i is connected to a c_j by an edge if and only if v_i is checked by c_j.

The decoding process is carried out by an iterative message-passing algorithm called "Belief Propagation Algorithm" [12]. In this algorithm, VN and CN iteratively exchange their soft-information to qualify the likelihood of the variable in accordance with the associated parity-check equation [1].

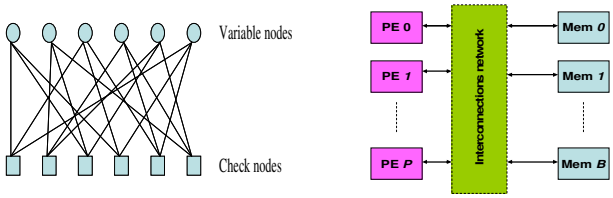
In literature, currently three main families of LDPC decoder architecture have been proposed:

- Fully-Parallel decoder architectures
- Serial decoder architectures
- Partially-Parallel decoder architectures

Fully-Parallel decoders suffer from prohibitive area and serial decoders from low throughput. Thus the only LDPC decoder architecture fulfilling the need of current communication standards is partially-parallel architecture. In partially-parallel architecture several processing elements *PE*s are used and set of variable nodes and set of check nodes are allotted to each PE. High throughput requirement can be achieved using a proper number of PEs, while the interconnection network cost tends to be less critical as compared to fully-parallel implementation. Typical architecture for partially-parallel decoder is shown in Figure 1.b in which P PEs are always connected with B memory banks where $P = B$. While designing partially-parallel decoder architecture, the implementation issues mainly arise due to the communication structure between VNs and CNs whereas the computation at variable node and check node is quite simple. Moreover, partially-parallel architecture suffers from memory accesses collision problem i.e., more than one PE concurrently accesses the same memory bank to read or write data. Hence, the communication structure becomes more and more challenging to design with the increase in the number of nodes, the number of iterations and the parallelism.

In this paper, we present a memory mapping methodology based on tripartite graph which is able to provide all the PEs conflict free parallel access to the memory banks. This algorithm provides conflict free memory mapping for all types of decoding methods, codeword lengths, code types and code rates.

The remainder of the paper is organized as follows. Section 2 presents a state of the art related to partially-parallel LDPC decoder design. Section 3 introduces the mapping problem. Section 4 defines concepts related to graph in order to understand the proposed approach. Section 5 details the mapping algorithm we propose. Finally, section 6 explains the algorithm through a pedagogical example.



a. Tanner graph of an LDPC b. Decoder architecture

Figure 1: LDPC code and decoder architecture

2. RELATED WORKS

Currently three classes of approaches to design partially-parallel LDPC decoder architectures exist to tackle the collision problem:

- Design LDPC codes to avoid collision problem [6], [7],
- Use extra memory elements and control logic in the interconnection network in order to remove conflicts [8], [9], [10],
- Find a memory mapping to provide conflict free access to all the memory banks at any time instance [11], [15], [16].

In the first category of decoder implementation, structured or architecture oriented LDPC codes are designed in order to avoid conflicts in accessing data from memory banks. These codes remove the memory access conflicts and simplify the interconnection network through the use of a barrel shifter [6] or a customized network [7]. However, constraints in the development of structured LDPC codes may cause degradation in code performance.

In the second class of decoder implementation, memory access conflicts are removed either through the addition of extra memory elements or complex interconnection network or both. In [8], configuration memories are used along with 2D-mesh network for LDPC codes of different block size and code rates. In [9], concurrent accesses to the same memory bank are avoided through the use of heterogeneous network. However, this network becomes complex with increasing degree of parallelization and suffers from reduction in the achievable throughput. In [10], binary de Bruijn network is employed for providing flexible on-chip network for LDPC decoder. Concurrent accesses to the same memory bank are avoided through dedicated routing algorithm which deflects one of the conflicted packets at the router. The flexibility in these complex interconnection networks is paid through additional hardware, increased decoding latency and power consumption.

In the last class, methodologies for solving collision problem are proposed to map the data in different memory banks for conflict free concurrent read/write accesses. In [11], the authors propose to use a mapping algorithm to remove memory conflicts in flexible LDPC decoders. However, the proposed approach is based on a simulated-annealing algorithm, so the user cannot predict when the algorithm will end. Moreover, it fails to optimize either the storage elements or the interconnection network. Finally, different heuristics [15], [16] have been proposed to solve the mapping problem in turbo and LDPC decoding. However, they consider in-place memory accesses in which data have to be read from and write to the same memory location.

Finally, conflict graph can be used. In this model, a node represents a data and two nodes are connected if and only if the associated data are accessed at the same time. Node coloring approach can then be used to solve the mapping problem: each color corresponds to one memory bank. Unfortunately only one color can be assigned to one node, i.e. a data can be stored in only one memory bank. This constraint may require more memory banks than needed (see [17] for more details). Similarly, number of algorithms have been proposed for coloring the edges of a bipartite graph by constructing partitions ([13] and [14] for example). Unfortunately, like node coloring approaches they can not be used to solve the mapping problem because each data is supposed to be

stored in one memory bank only i.e. only one color can be assigned to one edge.

3. PROBLEM FORMULATION

To explain the problem, we consider a set of K data $\{l_1, l2,..., l_K\}$ and a set of P processing elements $\{PE_1, PE_2,..., PE_P\}$ which iteratively process these K data in N time instances $\{t_1, t_2,..., t_N\}$.

In order to store these K data and to achieve parallel iterative processing for high throughput, a set of B memory banks $\{b_0, b_1,..., b_{B-1}\}$, where $B = P$, is used. All the memory banks have the same size M which is equal to $M = K/P$.

Mapping problem

Store K data in B memory banks in such a manner that P processing elements can, at each time instance, access B memory banks in parallel for first reading P data and then writing back these P data without any conflict.

To explain the problem, we introduce a mapping matrix in which we have P rows, related to the processing elements, and N columns, related to the time instances t_i. Each column is further divided into three sub-columns. First sub-column shows the data which need to be accessed in parallel by P processing elements at t_i whereas second sub-column contains the memory banks from where data are read and third sub-column represents the memory banks in which these data are written at t_i. Also, data in each row are processed by the processing element connected with this row. Figure 2 represents the mapping matrix in which we have $K = 6$, $P = B = 3$, $M = 2$ and $N = 6$. Each data is processed 3 times which shows the iterative nature of the data access. However, data accesses are interleaved in time and there is no regularity in processing the data; e.g., data 3 is successively processed in time instances t_1 and t_2 whereas the first access to the data 4 occurs at time instance t_3.

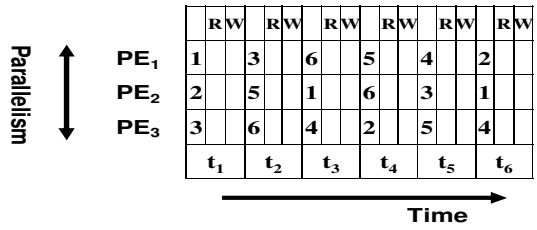


Figure 2: Mapping Matrix

Memory Mapping Constraints

To successfully map the data (i.e. to allow conflict free parallel memory access) (1) in a given number of memory banks and (2) to tackle the iterative nature of data access in error correction coding, the mapping matrix must fulfill the two following constraints:

1- At each time instance, all the memory banks have to be used one and only one time.
2- The bank of the last write access to a data must be the same as the bank of its first read access.

To tackle the mapping problem, we introduce the concept of multiple read and multiple write access. Therefore, we can access data in two ways: we can either (1) read and write a data from the same memory bank (if it is possible) i.e. like in classical approaches (see [15], [16] for example) or (2) read a data from one memory bank and then write it in a different one as we propose in this paper. The proposed approach, which allows to access data with an in-place strategy, is based on edge coloring of tripartite graph and is presented in section 5.

4. DEFINITIONS

A *graph* $G = (V, E)$ is a collection of nodes, set V, and edges, set E. If $v,w \in V$ then an edge $e(v,w) \in E$ is *incident* to v and to w, and vertices v and w are said *adjacent*. A *subgraph* of G is a graph whose vertices and edges are in G.

To *delete edge* (v,w) from G means to form the subgraph $G - (v,w)$, consisting of all vertices of G and all edges of G except (v,w).

A graph $G = (S_1 \cup S_2 \cup S_3, E)$ is *tripartite*, if *a set of graph vertices decomposed into three disjoint sets such that no two graph vertices within the same set are adjacent* i.e. $S_1 \cap S_2 \cap S_3 = \varnothing$.

The *degree* of vertex v is the number of edges incident to v. A graph is *regular* if all vertices have the same degree. A graph is *semi regular,* if all the vertices in any of its vertex set have the same degree.

A *path P* is a sequence of edges (v_1, v_2), (v_2, v_3),..., (v_{n-1}, v_n). The ends of P are vertices v_1 and v_n. If $v_1 \neq v_n$, P is *open*; otherwise P is *closed*. A graph is *connected* if there is a path between any two distinct vertices.

If S_i is the vertex set whose all the vertices have the same degree in a semi regular tripartite graph $G = (S_1 \cup S_2 \cup S_3, E)$ then *partition* in G is defined as a subgraph containing all the elements of S_i, (i.e. S_1, S_2 or S_3)

Lemma 1: When the degree d_t of a vertex of S_i in a *semi regular* graph is even then we have $d_t/2$ partitions in which each vertex's degree d_t' is 2.

Lemma 2: When the degree d_t of a vertex of S_i in a *semi regular* graph is odd then we have $\lfloor d_t/2 \rfloor$ partitions in which each vertex's degree d_t' is 2 and one subgraph in which d_t' is 1.

We finally define a *regular partition* in semi regular tripartite graph as a partition that respects either Lemma 1 or Lemma 2.

An edge coloring of G is an assignment of a color to each edge in G. An *edge chromatic number*, $\chi'(G)$, is the fewest number of colors necessary to color each edge of a graph so that no two edges incident to the same vertex have the same color.

5. PROPOSED APPROACH

The proposed approach is divided into two parts. In the first part, we model our problem as a tripartite graph based on mapping matrix i.e., the interleaving law. In the second part, we apply a 2-step coloring approach on the tripartite graph to color its edges so that data can be read from and be written to the memory without any conflict at any time instance.

5.1 Modeling

A Tripartite graph $G = (T_R \cup T_W \cup L, E)$ is constructed based on mapping matrix (e.g. Figure 3). Vertex sets T_R and T_W represent all the time instances at which data are read and written respectively. Vertex set L represents all the data used in the computation. An edge (l, t_{aR}) is incident to the data vertex l and to the read access time instance vertex t_{aR} if l needs to be read at t_{aR} . Similarly, an edge (t_{cW}, l) is incident to l and to the write access time instance vertex t_{cW} if l needs to be written at t_{cW}. Moreover, at each data vertex l, edges (l, t_{aR}) and (t_{cW}, l) are placed on two different sides of l as shown in Figure 3.b.

In order to follow the mapping constraint and for functional correctness of data accesses, the memory bank from which data is read from its current access must be the same as the memory bank in which the data is written in its previous access. If i is the access order of data l and n is the total number of times the data l is accessed, then $i = \{1, 2,....., n\}$.

Definition: two edges (l, t_{aR}) and (t_{cW}, l) are called *related edges* if

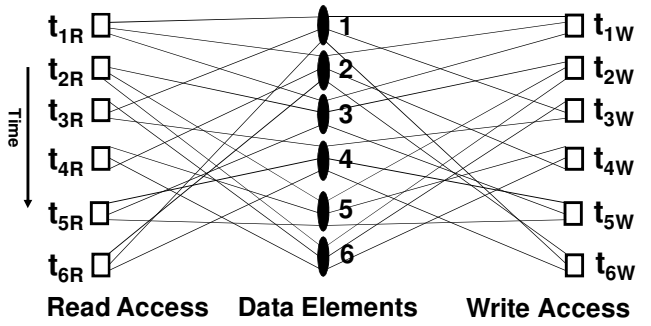
$$i = j - 1 \text{ for } i > 1$$
$$n \quad \text{for } i = 1$$

where $i = Order(l, t_{aR})$, $j = Order(t_{cW}, l)$ and where $Order(l, t_{aR})$ *and* $Order(t_{cw}, l)$ are respectively the read and the write access order of data l.

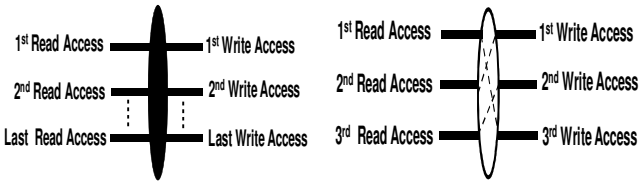
If colors of edges represent memory banks (as shown in *section 5.2*), then at each data vertex l, related edges must have the same color.

Related edges representation of data node l for $i = 3$ is shown in Figure 3.c. Related edges are connected with dotted line.

One interesting property of parallel LDPC decoding architecture is that the number of accesses to data or processing elements at any time instance is always equal which implies that corresponding tripartite graph is always semi regular at vertex set T_R and T_W. This implies that all the time nodes (either for read or write accesses) in the tripartite graph have the same degree d_t=P. Since vertex set T_R and T_W are always semi regular, the regular partitions contain all the vertices of both T_R and T_W with the degree requirement mentioned in *Lemma 1* and *2*.



a. Tripartite Graph of Figure. 2.



b. Data Node *l* Representation

c. Related edges representation for $i = 3$ of l

Figure 3: Tripartite Graph Representation

5.2 2-step coloring approach

A 2-step algorithm is used to color the edges of tripartite graph and hence to find a conflict free memory mapping: (1) tripartite graph is divided into *regular partitions* and then (2) each *regular partition* is colored independently.

Step I, Partitioning Algorithm:

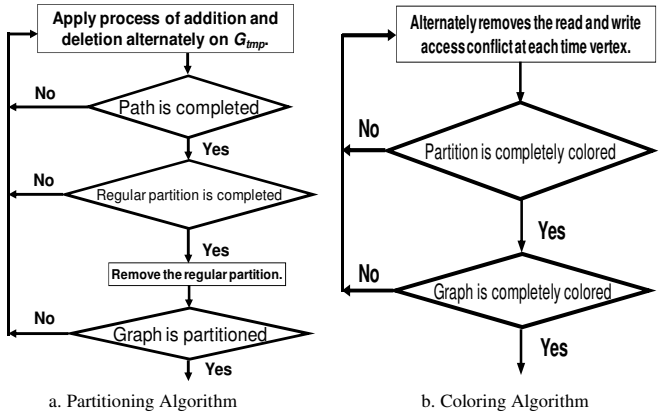
The algorithm (see Figure 4.a) begins constructing a path p_{curr} by selecting a data vertex $l_{curr.}$ *Process of addition* is then invoked to add an edge (l_{curr}, t_{aR}) into p_{curr} and its *related edge* (t_{cW}, l_{curr}) into the current set of related edges $Relp_{curr}$. *Process of deletion* is then applied to remove from G_{tmp} all *invalid* and their related edges from time nodes that belong to p_{curr}.

Definition: An edge is *invalid* if its selection decreases the number of edges at any read or write access vertex to less than d_t'. Otherwise, it is a *valid edge*.

Process of addition is again invoked at $t_{curr,R}$ to add another *valid edge* (t_{aR}, l_{next}) into p_{curr} and to reach at l_{next}. The related edge of (t_{aR}, l_{next}) is included in $Relp_{curr}$. Process of deletion is then applied. At that point $p_{curr} = \{(l_{curr}, t_{aR}), (t_{aR}, l_{next})\}$. The algorithm continues by alternating applying process of addition and process of deletion until p_{curr} is completed, i.e. the process of addition does not find any valid edge to be included in p_{curr}. At that point, p_{curr} and $Relp_{curr}$ are included in the current partition $Par_{curr.}$

While the partition is not regular (i.e. degree of valid edges at each read and write access time nodes is exactly d_t'), the algorithm starts constructing another path by using the remaining edges of G_{tmp}.

The algorithm starts constructing another partition on the remaining graph $G_{tmp} = G - Par_{cur}$ until G_{tmp} is not empty.



a. Partitioning Algorithm

b. Coloring Algorithm

Figure 4: Partitioning and Coloring Algorithms

Step II, Coloring Algorithm:

Our coloring algorithm (which flow chart is shown in Figure 4.b), colors each partition with at most two colors thanks to the construction of regular partitions.

For each partition Par_{curr}, the algorithm starts by choosing any read access time vertex $t_{init,R}$ whose edges are still not colored. First color is given to edge $(t_{init,R}, l)$ and its related edge $(l, t_{curr,W})$. Since at most two edges exist at each time vertex thanks to the construction of regular partitions, the algorithm gives a second color to $(t_{curr,W}, l_{next})$ and its related edge $(l_{next}, t_{curr,R})$. After reaching at $t_{curr,R}$, the algorithm tests whether $t_{curr,R} = t_{init,R}$. If not, then algorithm gives first color to the 2nd edge at $t_{curr,R}$ and its related edge until algorithm reaches at $t_{init,R}$. In case algorithm reaches $t_{init,R}$, it tests whether partition is completely colored. If not, then algorithm chooses another node as $t_{init,R}$ and repeats the same process until Par_{curr} is completely colored.

Partitioning and coloring algorithms are explained through a pedagogical example in the next section.

6. PRACTICAL IMPLEMENTATION

Let us present an example based on the mapping matrix in Figure 2. The first step is the construction of tripartite graph which is already depicted in Figure 3.a. This semi regular tripartite graph has each time vertex with degree $d_t = 3$. Following Lemma 2, we will have after applying the partitioning algorithm two partitions: one partition in which each time vertex's degree d_t' is 2 and one partition in which d_t' is 1.

The algorithm starts by selecting *data 1* and then invokes the process of addition which adds the first available edge $(1, t_{1R})$ into the path p_1, leading to $p_1 = \{(1, t_{1R})\}$. The related edge of $(1, t_{1R})$ that is $(1, t_{6w})$ is also included in the current set $Relp_1$. The selected read access edges and their related edges are represented by bold lines in Figure 5.a. The process of deletion is invoked but no invalid edge is found to be deleted. The process of addition continues by adding the edge $(t_{1R}, 2)$ into the path p_1 and its related edge $(2, t_{6W})$ into $Relp_1$. We have now $p_1 = \{(1, t_{1R}), (t_{1R}, 2)\}$ and $Relp_1 = \{(1, t_{6w}), (2, t_{6W})\}$. At this point, the number of access at t_{1R} and t_{6W} increases to 2 and the other unselected edges at t_{1R} and t_{6W} becomes invalid edges. So the process of deletion removes the invalid edge $(4, t_{6W})$ and its related edge $(4, t_{3R})$ as shown in Figure 5.b. The algorithm continues by alternately invoking the two processes until the path p_1 reaches at t_{6R}. We have at that point $p_1 = \{(1, t_{1R}), (t_{1R}, 2), (2, t_{4R}), (t_{4R}, 6), (6, t_{3R}), (t_{3R}, 1), (1, t_{6R})\}$ and $Relp_1 = \{(1, t_{6w}), (2, t_{6W}), (2, t_{1W}), (6, t_{3W}), (6, t_{2W}), (1, t_{1w}), (1, t_{3w})\}$. The edges of p_1 and $Relp_1$ are shown in Figure 5.c. At this point, the process of addition can choose either $(t_{6R}, 2)$ or $(t_{6R}, 4)$ to augment p_1. But choosing $(t_{6R}, 2)$ makes $(t_{6R}, 4)$ and its related edge $(4, t_{5W})$ invalid because the number of edges at t_{5W} becomes less than 2. So the process adds $(t_{6R}, 4)$ (the only available valid edge) into p_1, $(4, t_{5W})$ *into* $Relp_1$ and declares $(t_{6R}, 2)$ and its related edge $(2, t_{4W})$ as invalid as shown in Figure 5.d.

At this stage, the algorithm finds that no more valid edges available at data vertex 4 to be added in $p_1 = \{(1, t_{1R}), (t_{1R}, 2), (2, t_{4R}), (t_{4R}, 6),(6, t_{3R}), (t_{3R}, 1), (1, t_{6R}), (t_{6R}, 4)\}$ and $Relp_1 = \{(1, t_{6w}), (2, t_{6W}), (2, t_{1W}), (6, t_{3W}), (6, t_{2W}), (1, t_{1w}), (1, t_{3w}), (4, t_{5W})\}$ as shown in Figure 6.a. The algorithm adds p_1 and $Relp_1$ into Par_1 but Par_1 does not form a regular partition because t_{2R} and t_{5R} are not included in p_1. So the algorithm starts to construct a new path p_2 by again invoking the process of addition and deletion. The resultant path $p_2 = \{(3, t_{5R}), (t_{5R}, 5), (5, t_{2R}), (t_{2R}, 6)\}$ is shown in Figure 6.b. Now the partition Par_1 is the union of all the paths and their related edge sets, $Par_1 = p_1 + p_2 + Relp_1 + Relp_2$. Again the algorithm tests whether Par_1 constitutes a regular partition. This time the test is successful and the Par_1 is declared as regular partition (see Figure 6.c).

After the construction of Par_1, the algorithm finds that the graph is not completely traversed. So the algorithm deletes Par_1 to obtain the graph $G_{tmp} = G - Par_1$ and applies again the processes on G_{tmp} to obtain the paths, $p'_1 = \{(2, t_{6R})\}$, $p'_2 = \{(3, t_{1R})\}$, $p'_3 = \{(3, t_{2R})\}$, $p'_4 = \{(4, t_{3R})\}$, $p'_5 = \{(4, t_{5R})\}$, $p'_6 = \{(5, t_{4R})\}$.

Similarly partition Par_2 is the sum of all the traversed paths and their related edges as given below,

$Par_2 = p'_1 + p'_2 + p'_3 + p'_4 + p'_5 + p'_6 + Relp'_1 + Relp'_2 + Relp'_3 + Relp'_4 + Relp'_5 + Relp'_6$ (see Figure 6.d).

After the construction of Par_2, the partitioning algorithm finds that the graph is completely traversed and is terminated.

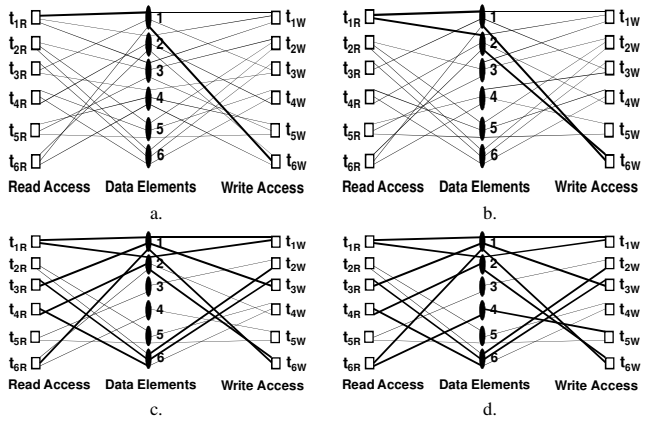


Figure 5*:* Path construction through Partitioning Algorithm

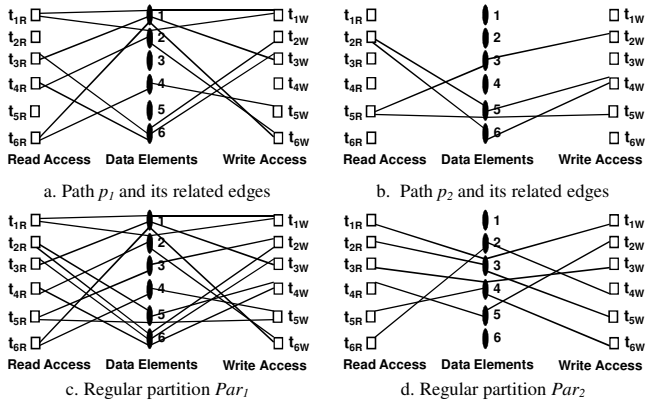


Figure 6: Path construction through Partitioning Algorithm

After the generation of the partitions, each partition is colored depending on the degree d_t' of its time node. For example, the Par_1 is colored with, d_t' = 2, colors and the Par_2 is colored with, d_t' = 1, color. To color the partition Par_1, the algorithm starts from any read access time vertex whose edges are still not colored. In this example, the algorithm begins from t_{1R} and gives one color b_0 to *(t_{1R} , 1)* and its related edge *(1, t_{6W})* to reach at t_{6W}. At t_{6W}, the algorithm then gives different color b_1 to the other edge *(t_{6W}, 2)* and its related edge *(t_{1R} , 2)* to reach at t_{1R} as shown in Figure 7.a. In this figure, grey straight line represents color b_0 and grey dotted line represents color b_1.

The algorithm finds that t_{1R} is completely colored so it chooses another uncolored read access time vertex t_{2R} and gives color b_0 to *(t_{2R} , 5)* and its related edge *(5, t_{5W})* to reach at t_{5W}. At t_{5W}, the algorithm gives different color b_1 to the other edge *(t_{5W} , 4)* and its related edge *(t_{6R}, 4)* to reach at t_{6R} as shown in Figure 7.b.

The algorithm continues until the partition is completely colored. The complete coloring of Par_1 is shown in Figure 8.a. The coloring of Par_2 is easier: all the edges are colored with one single color b_2 represented by grey big dotted lines in Figure 8.b because d_t' = 1 as already mentioned.

The complete coloring of G is shown in Figure 9.a. The corresponding mapping matrix is presented in Figure 9.b.

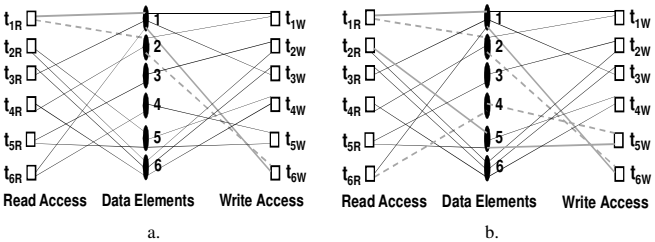


Figure 7: Conflict Free Edge Coloring of Par_1

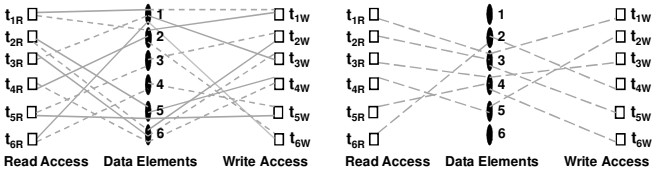
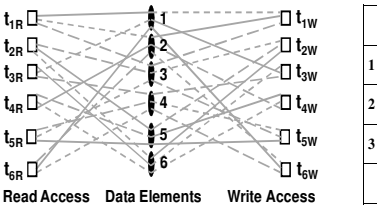


Figure 8: Conflict Free Edge Coloring of Par_1 and Par_2



a. Coloring of G

	R	W		R	W		R	W		R	W		R	W		R	W
1	b_0	b_1	3	b_2	b_1	6	b_0	b_1	5	b_2	b_0	4	b_2	b_1	2	b_2	b_1
2	b_1	b_0	5	b_0	b_2	1	b_1	b_0	6	b_1	b_1	3	b_1	b_2	1	b_0	b_0
3	b_2	b_2	6	b_1	b_0	4	b_2	b_2	2	b_0	b_2	5	b_0	b_0	4	b_1	b_2
	t_1			t_2			t_3			t_4			t_5			t_6	

b. Mapping matrix

Figure 9: Conflict free edge coloring of G and corresponding mapping matrix

7. CONCLUSION

In this paper, we proposed an approach to solve the *mapping problem* that arises when designing parallel LDPC interleaver. The concept of tripartite edge coloring has been introduced to design hardware architectures supporting multiple read/write accesses. This approach can be used to solve the mapping problems in other signal processing, communication domains... In future works, additional constraints will be added to take into account the complexity of interconnection network.

REFERENCE

[1] R.G. Gallager,"Low Density Parity Check Codes" *IRE Trans. Info. Theory, 21-28*, 1962.

[2] Tanner, R. M., 1981. "A recursive approach to low complexity codes". *IEEE Trans. Info. Theory, 533-547*

[3] "Frame structure channel coding and modulation for the second generation digital terrestrial television broadcasting system (DVB-T2)," *DVB DocumentA122,* 2008.

[4] IEEE 802.11n. "Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput", *IEEE P802.11n/D1.0*, 2006

[5] "Air interface for fixed and mobile broadband wireless access systems," *in P802.16e/D12 Draft*, (Washington, DC, USA), pp. 100-105, IEEE, 2005

[6] M.M. Mansour, N.R. Shanbhag, "High-throughput, LDPC decoders," *IEEE Trans. on Very Large Scale Integration VLSI Systems,* vol.11, pp.976-996, 2003.

[7] Y.Chen, D.Hocevar, "A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder". *in Global Telecommunication Conf., 113-117*, 2003.

[8] Theocharides, T., Link, G., Vijaykrishnan, N., and Irwin, M. J., 2005. "Implementing LDPC Decoding on a Network-on-Chip". *Proc. of the int. Conference on VLSI Design*.

[9] Kienle, F., Thul, M. J., and When, N., 2003. "Implementation Issues of Scalable LDPC-Decoders". *in Proceeding of 3rd International Symposium on Turbo Codes and Related Topics, Brest, France, 291-294.*

[10] H.Moussa, A.Baghdadi, M.Jezequel."Binary de Bruijn on-chip netwok for a flexible multiprocessor LDPC decoder".*45th ACM/IEEE DAC, p.429-434*, 2008.

[11] F.Quaglio, F.Vacca, C.Castellano, A.Tarable, M G.Asera. "Interconnection Framework for High-Throughput, Flexible LDPC Decoders". *In proceeding Design Automation and Test in Europe Conference and Exhibition*, 2006.

[12] J.Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible reference, *Morgan Kaufmann*, 1988.

[13] H.N. Gabow, "Using Euler partitions to edge color bipartite multigraphs", *International Journal of Computer and Information Sciences* 5 (1976) 345-355.

[14] R.Cole, J. Hopcroft, "On edge coloring bipartite graphs", *SIAM Journal on Computing* 11 (1982) 540-546.

[15] A.Tarable, S. Benedetto and G.Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures", IEEE Trans. Inf. Theory, vol.50, no.9, pp.2002-2009, Sep. 2004.

[16] C. Chavet, P. Coussy, P. Urard and E. Martin, "Static Address Generation Easing: a Design Methodology for Parallel Interleaver Architecture". *In proceeding ICASSP 2010.*

[17] C. Chavet, P. Coussy, "A memory Mapping Approach for Parallel Interleaver design with multiples read and write accesses". In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2010.